ns-3 Training

Session 1: Monday May 11

ns-3 Annual meeting
May 2015
Introduction and logistics

• CTTC facilities
• Meals and coffee
• Wi-Fi
• Wiki page:
• Meet your instructors
Monday agenda

• Monday
  – ns-3 survey and overview tutorial, starting from first principles and walking through the running of simulations, configuration management, architecture of the software core, network emulation, and development practices using ns-3.
  – Methodology and workflow for developing new models in ns-3, using a case study.
  – Several tools used to extract and visualize data from ns-3 simulations, including the flow monitor, network animator NetAnim, Python-based visualizer, and the ns-3 tracing system.
Tuesday agenda

• Tuesday
  – (09:00-10:30) Large-scale, distributed simulations with ns-3 (instructor: Peter Barnes)
  – (11:00-12:30) An introduction to the Direct Code Execution (DCE) environment, enabling users to use real application and Linux networking code in ns-3 (instructor: Hajime Tazaki)
  – Lunch break
  – (14:00-16:00) A survey of the LTE models, including model architecture, propagation models, LTE Radio Protocol Stack and EPC model. (instructor: Nicola Baldo)
  – 16:30-18:00) A tutorial on vehicular communication simulations, including mobility, WiFi and WAVE models, and propagation. (instructor: Konstantinos Katsaros)
Later in the week

- WNS3 Wednesday and Thursday morning
- ns-3 Consortium Annual Meeting (16h00 Thursday)
- Developer meetings Friday
ns-3 training goals

• Learn about the project scope, and where to get additional help
• Understand the architecture and design goals of the software
• Introduce how to write new code for the simulator
• Learn about selected topics in more detail
• Answer your questions
Motivations for ns-3 project

Develop an extensible simulation environment for networking research

1) a tool **aligned with the experimentation needs** of modern networking research

2) a tool that **elevates the technical rigor** of network simulation practice

3) an **open-source project** that encourages community contribution, peer review, and long-term maintenance and validation of the software
Network performance evaluation options

- ns-3 enables researchers to more easily move between simulations, test beds, and experiments.
**ns history**

1988: REAL (Keshav)

1990s: ns-1
    - 1990: ns-1
    - 1996: ns-2

1997-2000: DARPA VINT

2001-04: DARPA SAMAN, NSF CONSER

2006: NSF CISE CRI Awards

**Inputs:** yans, GTNetS, ns-2

ns-3 core development (2006-08)

June 2008: ns-3.1

May 2015: ns-3.23

Regular releases
Relationship to ns-2

ns-3 is a new simulator, without backward compatibility

Similarities to ns-2:
• C++ software core
• GNU GPLv2 licensing
• ported ns-2 models: random variables, error models, OLSR, Calendar Queue scheduler

Differences:
• Python scripting (or C++ programs) replaces OTcl
• most of the core rewritten
• new animators, configuration tools, etc. are in work
• ns-2 is no longer actively maintained/supported
How the project operates

- Project provides three annual software releases
- Users interact on mailing lists and using Bugzilla bug tracker
- Code may be proposed for merge
  - Code reviews occur on a Google site
- Maintainers (one for each module) fix or delegate bugs, participate in reviews
- Project has been conducting annual workshop and developer meeting around SIMUTools through 2013
  - Some additional meetings on ad hoc basis
- Google Summer of Code (March-August) six of the past seven summers
Sustainment

• The NS-3 Consortium is a collection of organizations cooperating to support and develop the ns-3 software.

• It operates in support of the open source project
  – by providing a point of contact between industrial members and ns-3 developers,
  – by sponsoring events in support of ns-3 such as users' days and workshops,
  – by guaranteeing maintenance support for ns-3's core, and
  – by supporting administrative activities necessary to conduct a large open source project.
A common question is "How many ns-3 papers are there?"

- Small survey of 139 paper results from 2013-14 search of IEEE library (top relevant results)
- Some papers matched multiple categories
- Hot topics:
  - LTE/cellular networks (15)
  - Wireless routing protocols (14)
  - Sensor networks (13)
  - Wireless MAC and PHY protocols (11)
## Paper counts by topic

<table>
<thead>
<tr>
<th>Topic</th>
<th>Count</th>
<th>Topic</th>
<th>Count</th>
</tr>
</thead>
<tbody>
<tr>
<td>LTE/Cellular</td>
<td>15</td>
<td>Network coding</td>
<td>4</td>
</tr>
<tr>
<td>Wireless routing protocols</td>
<td>14</td>
<td>Datacenter networks</td>
<td>4</td>
</tr>
<tr>
<td>Wireless sensor networks</td>
<td>13</td>
<td>Distributed systems</td>
<td>4</td>
</tr>
<tr>
<td>Wireless MAC/PHY</td>
<td>11</td>
<td>Optical links</td>
<td>3</td>
</tr>
<tr>
<td>Wireless QoS</td>
<td>9</td>
<td>Misc. physical links</td>
<td>3</td>
</tr>
<tr>
<td>Vehicular networks</td>
<td>9</td>
<td>Multicast</td>
<td>3</td>
</tr>
<tr>
<td>TCP/congestion control</td>
<td>9</td>
<td>Misc. security</td>
<td>2</td>
</tr>
<tr>
<td>Wireless security</td>
<td>9</td>
<td>Wired routers</td>
<td>2</td>
</tr>
<tr>
<td>About ns-3 itself</td>
<td>8</td>
<td>Wireless QoS</td>
<td>2</td>
</tr>
<tr>
<td>Wifi/mesh networks</td>
<td>7</td>
<td>WiMAX</td>
<td>1</td>
</tr>
<tr>
<td>Voice/video apps</td>
<td>6</td>
<td>Mobility</td>
<td>1</td>
</tr>
<tr>
<td>Energy/resource consumption</td>
<td>6</td>
<td>Misc. routing</td>
<td>1</td>
</tr>
<tr>
<td>DTN and space networks</td>
<td>5</td>
<td>Miscellaneous</td>
<td>1</td>
</tr>
<tr>
<td>Misc. wireless</td>
<td>5</td>
<td></td>
<td></td>
</tr>
</tbody>
</table>
Acknowledgment of support
• Software overview
Options for working along

1) Download the required packages onto your (Linux, OS X, or BSD) system
2) Download or copy the ISO image (Live DVD)
3) Browse the code online: https://code.nsnam.org
ns-3 main website

- Project home: [https://www.nsnam.org](https://www.nsnam.org)
Software overview

- ns-3 is written in C++, with bindings available for Python
  - simulation programs are C++ executables or Python programs
  - ~350,000 lines of C++ (estimate based on cloc source code analysis)
- ns-3 is a GNU GPLv2-licensed project
- ns-3 is mainly supported for Linux, OS X, and FreeBSD
  - Windows Visual Studio port available
- ns-3 is not backwards-compatible with ns-2
Discrete-event simulation basics

- Simulation time moves in discrete jumps from event to event
- C++ functions schedule events to occur at specific simulation times
- A simulation scheduler orders the event execution
- `Simulation::Run()` executes a single-threaded event list
- Simulation stops at specific time or when events end
The basic ns-3 architecture

Application

Protocol stack

Node

NetDevice

Sockets-like API

Packet(s)

Channel

Channel

Application

Protocol stack

Node

NetDevice
Software orientation

Key differences from other network simulators:

1) Command-line, Unix orientation
   – vs. Integrated Development Environment (IDE)

2) Simulations and models written directly in C++ and Python
   – vs. a domain-specific simulation language
Software organization

• Two levels of ns-3 software and libraries

1) Several supporting libraries, not system-installed, can be in parallel to ns-3

2) ns-3 modules exist within the ns-3 directory
Current models

devices
- bridge
- csma
- emu
- point-to-point
- wifi
- wimax

Node class
NetDevice ABC
Address types (IPv4, MAC, etc.)
Queues
Socket ABC
Ip4 ABCs
Packet sockets

Smart pointers
Dynamic types
Attributes
Callbacks
Tracing
Logging
Random Variables

applications
- internet (IPv4/v6)

protocols
- aodv
- dsdv
- olsr
- click
- openflow
- BRITE

energy

utilities
- config-store
- flow-monitor
- netanim
- stats
- topology-read
- mix-vector-routing
- stats

Packets
Packet Tags
Packet Headers
Pcap/ascii file writing

Events
Scheduler
Time arithmetic

network

core

NS-3 Introduction
July 2014
Module organization

- models/
- examples/
- tests/
- bindings/
- doc/
- wscript
ns-3 programs

- ns-3 programs are C++ executables that link the needed shared libraries
  - or Python programs that import the needed modules
- The ns-3 build tool, called 'waf', can be used to run programs
- waf will place headers, object files, libraries, and executables in a 'build' directory
Python bindings

• ns-3 uses a program called PyBindGen to generate Python bindings for all libraries

Diagram:

1. C++ header
2. Intermediate Python program
3. C++ bindings code
4. Python module

- (py)gccxml
- PyBindGen
- C++ compiler
Integrating other tools and libraries
Other libraries

• more sophisticated scenarios and models typically leverage other libraries
• ns-3 main distribution uses optional libraries (libxml2, gsl, mysql) but care is taken to avoid strict build dependencies
• the 'bake' tool (described later) helps to manage library dependencies
• users are free to write their own Makefiles or wscripts to do something special
Gnuplot

- src/tools/gnuplot.{cc,h}
- C++ wrapper around gnuplot
- classes:
  - Gnuplot
  - GnuplotDataset
    - Gnuplot2dDataset, Gnuplot2dFunction
    - Gnuplot3dDataset, Gnuplot3dFunction
Enabling gnuplot for your code

- examples/wireless/wifi-clear-channel-cmu.cc

```cpp
CommandLine cmd;
cmd.Parse (argc, argv);

Gnuplot gnuplot = Gnuplot ("clear-channel.eps");

for (uint32_t i = 0; i < modes.size (); i++)
{
    std::cout << modes[i] << std::endl;
    Gnuplot2dDataset dataset (modes[i]);
}
```

produce a plot file that will generate an EPS figure

one dataset per mode

```cpp
uint32_t pktsRecvd = experiment.Run (wifi, wifiPhy, wifiMac, wifiChannel);

dataset.Add (rss, pktsRecvd);
```

Add data to dataset

```cpp
gnuplot.AddDataset (dataset);
```

Add dataset to plot
Matplotlib

- `src/core/examples/sample-rng-plot.py`

```python
# Demonstrate use of ns-3 as a random number generator integrated
# with matplotlib's plotting tools; adapted from Gustavo Carneiro's ns-3 tutorial

import numpy as np
import matplotlib.pyplot as plt
import ns.core

# mu, var = 100, 225
rng = ns.core.NormalVariable(100.0, 225.0)
x = [rng.GetValue() for t in range(10000)]

# the histogram of the data
n, bins, patches = plt.hist(x, 50, normed=1, facecolor='g', alpha=0.75)
plt.title('ns-3 histogram')
plt.text(60, .025, r'$\mu=100, \sigma=15$')
plt.axis([40, 160, 0, 0.03])
plt.grid(True)
plt.show()
```
Click Modular Router

The Click Modular Router Project

NEWS (September 24, 2011): Click 2.0.1 released!

Symposium on Click Modular Router was November 23-24, 2009, Ghent, Belgium! An excellent time was had. Video of the presentations is now available.

This is the DokuWiki for the Click modular router. Click was originally developed at MIT with subsequent development at Mazu Networks, ICIR, UCLA, and Meraki.
mininet emulator

Link modeling using ns 3

Contents

- Introduction
  - ns-3 emulation features
  - Link simulation with ns-3
- Details
  - How to achieve communication of ns-3 process with TAP interfaces in distinct namespaces?
  - Architecture: single ns-3 thread or multiple processes?
- Code
  - Mininet
  - ns-3 patches
Co-simulation frameworks have emerged

- PNNL's FNCS framework integrates ns-3 with transmission and distribution simulators

FAQs

• Does ns-3 have a Windows version?
  – Yes, for Visual Studio 2012

• Does ns-3 support Eclipse or other IDEs?
  – Instructions have been contributed by users

• Is ns-3 provided in Linux or OS X package systems (e.g. Debian packages)?
  – Ubuntu/Debian packages for ns-3.17 release

• Does ns-3 support NRL protolib applications?
  – Not yet
Summarizing

- ns-3 models are written in C++ and compiled into libraries
  - Python bindings are optionally created
- ns-3 programs are C++ executables or Python programs that call the ns-3 public API and can call other libraries
- ns-3 is oriented towards the command-line
- ns-3 uses no domain specific language
- ns-3 is not compatible with ns-2
Finding documentation and code
Resources

Web site:
http://www.nsnam.org

Mailing lists:
https://groups.google.com/forum/#!forum/ns-3-users
http://mailman.isi.edu/mailman/listinfo/ns-developers

Wiki:
http://www.nsnam.org/wiki/

Tutorial:
http://www.nsnam.org/docs/tutorial/tutorial.html

IRC: #ns-3 at freenode.net
Suggested steps

• Work through the ns-3 tutorial
• Browse the source code and other project documentation
  – manual, model library, Doxygen, wiki
  – ns-3 Consortium tutorials (May 2014)
    • https://www.nsnam.org/consortium/activities/training/
• Ask on ns-3-users mailing list if you still have questions
  – We try to answer most questions
APIs

- Most of the ns-3 API is documented with Doxygen
  - [https://www.nsnam.org/doxygen](https://www.nsnam.org/doxygen)
Contributed code and associated projects
Reading existing code

• Much insight can be gained from reading ns-3 examples and tests, and running them yourselves

• Many core features of ns-3 are only demonstrated in the core test suite (src/core/test)

• Stepping through code with a debugger is informative
  – callbacks and templates make it more challenging than usual
ns-3 build systems
Software introduction

• Download the latest release
  – tar xjf ns-allinone-3.19.tar.bz2

• Clone the latest development code
  – hg clone http://code.nsnam.org/ns-3-allinone

Q. What is "hg clone"?
A. Mercurial (http://www.selenic.com) is our source code control tool.
Software building

• Two levels of ns-3 build

1) **bake** (a Python-based build system to control an ordered build of ns-3 and its libraries)

2) **waf**, a build system written in Python

3) **build.py** (a custom Python build script to control an ordered build of ns-3 and its libraries)  
   
   **--- may eventually be deprecated**
ns-3 uses the 'waf' build system

• Waf is a Python-based framework for configuring, compiling and installing applications.
  – It is a replacement for other tools such as Autotools, Scons, CMake or Ant

• For those familiar with autotools:
  • configure ➔ ./waf configure
  • make ➔ ./waf build
waf configuration

• Key waf configuration examples

  ./waf configure
  --enable-examples
  --enable-tests
  --disable-python
  --enable-modules

• Whenever build scripts change, need to reconfigure

Demo: ./waf --help
  ./waf configure --enable-examples --enable-tests --enable-modules='core'

Look at: build/c4che/_cache.py
```python
def build(bld):
    obj = bld.create_ns3_module('csma', ['network', 'applications'])
    obj.source = [
        'model/backoff.cc',
        'model/csma-net-device.cc',
        'model/csma-channel.cc',
        'helper/csma-helper.cc',
    ]
    headers = bld.new_task_gen(features=['ns3header'])
    headers.module = 'csma'
    headers.source = [
        'model/backoff.h',
        'model/csma-net-device.h',
        'model/csma-channel.h',
        'helper/csma-helper.h',
    ]

    if bld.env['ENABLE_EXAMPLES']:
        bld.add_subdirs('examples')

    bld.ns3_python_bindings()
```
waf build

- Once project is configured, can build via 
  
  ./waf build or ./waf

- waf will build in parallel on multiple cores
- waf displays modules built at end of build

Demo: ./waf build

Look at: build/ libraries and executables
Running programs

- `./waf shell` provides a special shell for running programs
  - Sets key environment variables

  ./waf --run sample-simulator
  ./waf --pyrun src/core/examples/sample-simulator.py
Build variations

- Configuring a build type is done at waf configuration time
- debug build (default): all asserts and debugging code enabled
  
  ./waf -d debug configure

- optimized
  
  ./waf -d optimized configure

- static libraries
  
  ./waf --enable-static configure
Controlling the modular build

- One way to disable modules:
  - ./waf configure --enable-modules='a','b','c'

- The .ns3rc file (found in utils/ directory) can be used to control the modules built

- Precedence in controlling build
  1) command line arguments
  2) .ns3rc in ns-3 top level directory
  3) .ns3rc in user's home directory

Demo how .ns3rc works
Building without wscript

• The scratch/ directory can be used to build programs without wscripts

Demo how programs can be built without wscripts
bake overview

- Open source project maintains a (more stable) core
- Models migrate to a more federated development process

"bake" tool (Lacage and Camara)

Components:
- build client
- "module store" server
- module metadata

Figure source: Daniel Camara
bake basics

- bake can be used to build the Python bindings toolchain, Direct Code Execution, Network Simulation Cradle, etc.

```
./bake.py configure -e <module>
./bake.py show
./bake.py download
./bake.py build
```
Demo: ./waf build

Look at: build/ libraries and executables
Visualization
PyViz overview

- Developed by Gustavo Carneiro
- Live simulation visualizer (no trace files)
- Useful for debugging
  - mobility model behavior
  - where are packets being dropped?
- Built-in interactive Python console to debug the state of running objects
- Works with Python and C++ programs
Pyviz screenshot (Graphviz layout)
Pyviz and FlowMonitor

- src/flow-monitor/examples/wifi-olsr-flowmon.py
Enabling PyViz in your simulations

• Make sure PyViz is enabled in the build

![Build output]

• If program supports CommandLine parsing, pass the option
  ```
  --SimulatorImplementationType=ns3::Visual Simulator Impl
  ```

• Alternatively, pass the "--vis" option
FlowMonitor

- Network monitoring framework found in `src/flow-monitor/`

- **Goals:**
  - detect all flows passing through network
  - stores metrics for analysis such as bitrates, duration, delays, packet sizes, packet loss ratios

FlowMonitor architecture

• Basic classes
  – FlowMonitor
  – FlowProbe
  – FlowClassifier
  – FlowMonitorHelper

• IPv6 coming in ns-3.20 release

FlowMonitor statistics

- Statistics gathered

```
FlowMonitor::FlowStats
+timeFirstTxPacket: Time
+timeFirstRxPacket: Time
+timeLastTxPacket: Time
+timeLastRxPacket: Time
+delaySum: Time
+jitterSum: Time
+txBytes: uint64_t
+rxBytes: uint64_t
+txPackets: uint32_t
+rxPackets: uint32_t
+lostPackets: uint32_t
+timesForwarded: uint32_t
+delayHistogram: Histogram
+jitterHistogram: Histogram
+packetSizeHistogram: Histogram
+packetsDropped: std::vector<uint32_t>
+bytesDropped: std::vector<uint64_t>
```

```
FlowProbe::FlowStats
+delayFromFirstProbeSum: Time
+bytes: uint64_t
+packets: uint32_t
+packetsDropped: std::vector<uint32_t>
+bytesDropped: std::vector<uint64_t>
```

```
Histogram
+Histogram(binWidth:double)
+GetNBins(): uint32_t const
+GetBinStart(index:uint32_t): double const
+GetBinEnd(): double const
+GetBinWidth(index:uint32_t): double const
+SetDefaultBinWidth(width:double)
+GetBinCount(index:uint32_t): uint32_t
+AddValue(value:double)
```
FlowMonitor configuration

- example/wireless/wifi-hidden-terminal.cc

```cpp
// 8. Install FlowMonitor on all nodes
FlowMonitorHelper flowmon;
Ptr<FlowMonitor> monitor = flowmon.InstallAll();

// 9. Run simulation for 10 seconds
Simulator::Stop (Seconds (10));
Simulator::Run ();

// 10. Print per flow statistics
monitor->CheckForLostPackets ();
Ptr<Ipv4FlowClassifier> classifier = DynamicCast<Ipv4FlowClassifier> (flowmon.GetClassifier ());
std::map<FlowId, FlowMonitor::FlowStats> stats = monitor->GetFlowStats ();
for (std::map<FlowId, FlowMonitor::FlowStats>::const_iterator i = stats.begin (); i != stats.end (); ++i)
{
    // first 2 FlowIds are for ECHO apps, we don't want to display them
    if (i->first > 2)
    {
        Ipv4FlowClassifier::FiveTuple t = classifier->FindFlow (i->first);
        std::cout << "Flow " << i->first - 2 << " " << t.sourceAddress << " -> " << t.destinationAddress << "\n";
        std::cout << " Tx Bytes: " << i->second.txBytes << "\n";
        std::cout << " Rx Bytes: " << i->second.rxBytes << "\n";
        std::cout << " Throughput: " << i->second.rxBytes * 8.0 / 10.0 / 1024 / 1024 << " Mbps\n";
    }
}
```
FlowMonitor output

- This program exports statistics to stdout
- Other examples integrate with PyViz

```
Hidden station experiment with RTS/CTS disabled:
Flow 1 (10.0.0.1 -> 10.0.0.2)
   Tx Bytes: 3847500
   Rx Bytes: 316464
   Throughput: 0.241443 Mbps
Flow 2 (10.0.0.3 -> 10.0.0.2)
   Tx Bytes: 3848412
   Rx Bytes: 336756
   Throughput: 0.256924 Mbps

Hidden station experiment with RTS/CTS enabled:
Flow 1 (10.0.0.1 -> 10.0.0.2)
   Tx Bytes: 3847500
   Rx Bytes: 386660
   Throughput: 0.233963 Mbps
Flow 2 (10.0.0.3 -> 10.0.0.2)
   Tx Bytes: 3848412
   Rx Bytes: 274740
   Throughput: 0.20961 Mbps
```
NetAnim

- "NetAnim" by George Riley and John Abraham
NetAnim key features

- Animate packets over wired-links and wireless-links
  - limited support for LTE traces
- Packet timeline with regex filter on packet meta-data.
- Node position statistics with node trajectory plotting (path of a mobile node).
- Print brief packet-meta data on packets
Placeholder for netanim videos